AI on Classic Video Games Using Reinforcement Learning

By

Shivika Sodhi

Project Advisor:

Dr. Chris Pollett

Committee Members:

Dr Jenny Lam

Dr. Robert Chun

SJSU SAN JOSÉ STATE UNIVERSITY

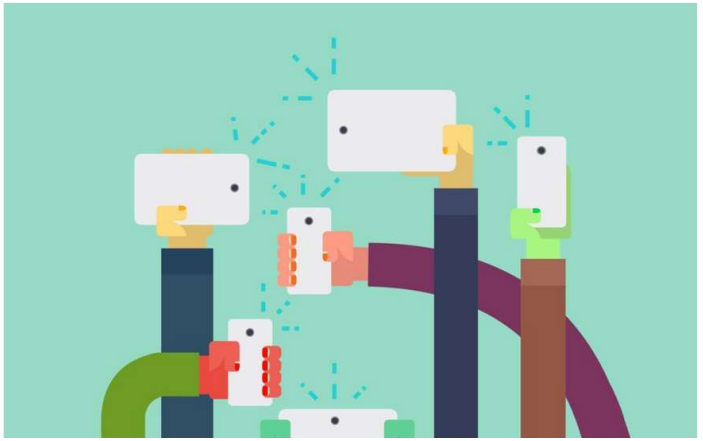
Outline

- Introduction
- Archon
- Related Work
- Reinforcement Learning
  - Q-learning Algorithm
- Deep Learning
  - Convolutional Neural Network Algorithm
- Approach 1
- Approach 2
- Conclusion, Result, Future Work, References

The Big Picture

Bixby

GOOGLE LENS TURNS YOUR CAMERA INTO A SEARCH BOX



Chatbot

Google wants to use Tensorflow software to teach machines how to compose music and create works of art. DeepDream, Google's visual AI that could transform photos into psychedelic art (such as Salvador Dali's Persistence of Memory, pictured), worked on a similar principle

SJSU SAN JOSÉ STATE UNIVERSITY

Introduction

Project Goals

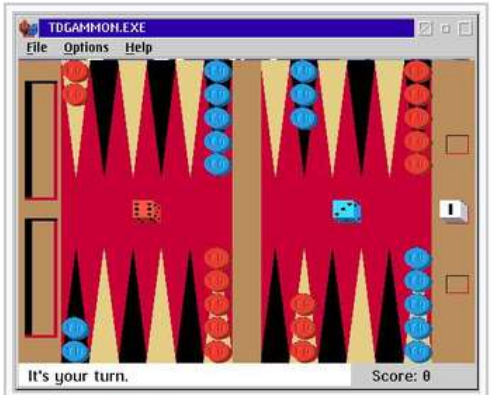
Primary Goal:

- Design an artificially intelligent bot to play a classic video game, Archon: the light and dark.
- The agent learns on its own to automatically determine successful strategies that lead to the greatest long-term rewards.

Related Work

TD Gammon

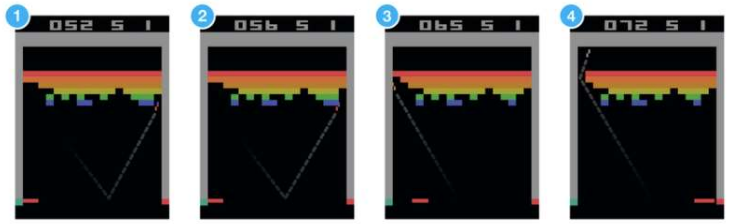
- Computer backgammon program
- Developed in 1992 by Gerald Tesauro at IBM
- Achieved a level of play just slightly below that of the top human backgammon players of the time



OS/2 TD-Gammon game screenshot

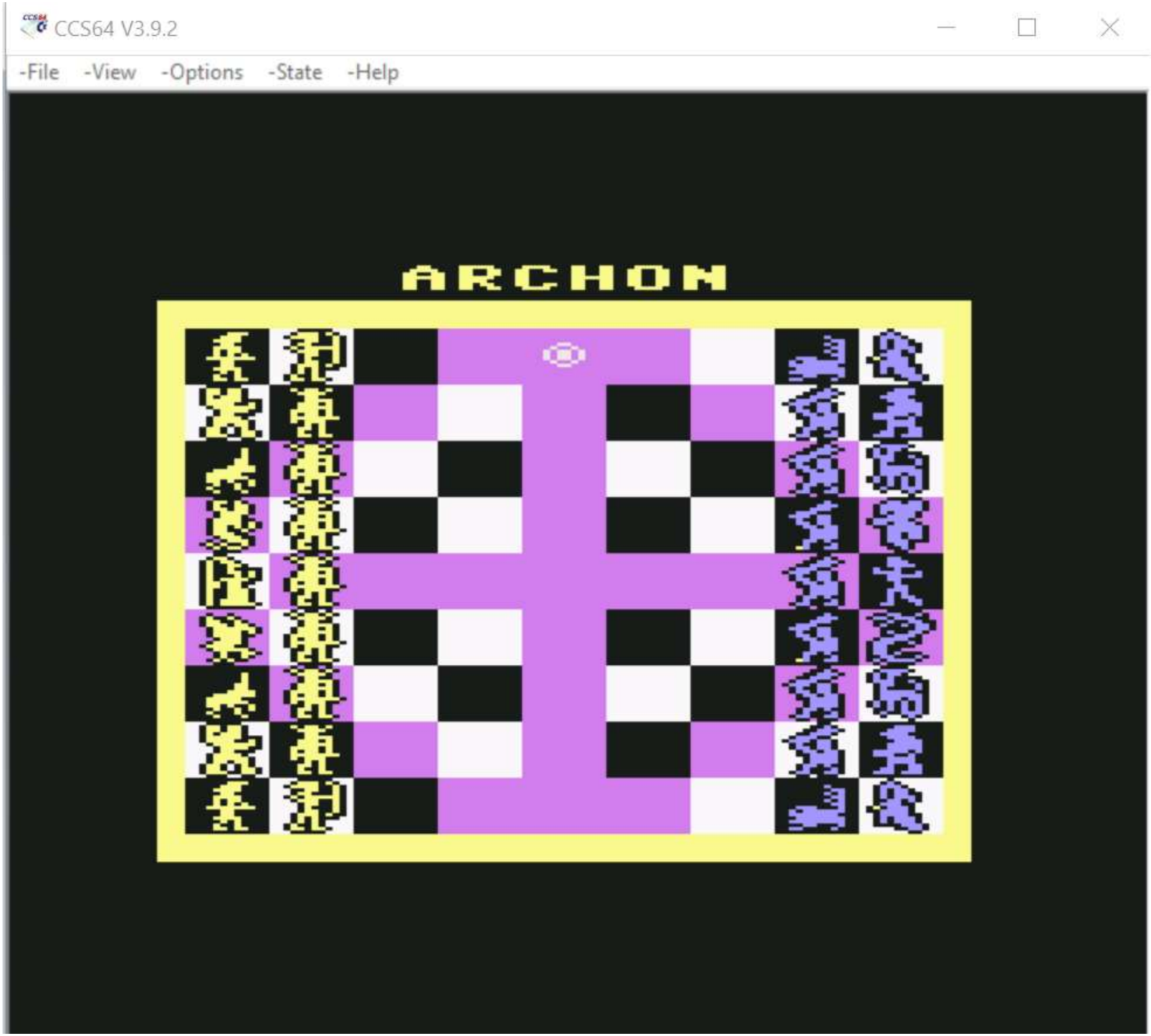
SJSU SAN JOSÉ STATE UNIVERSITY

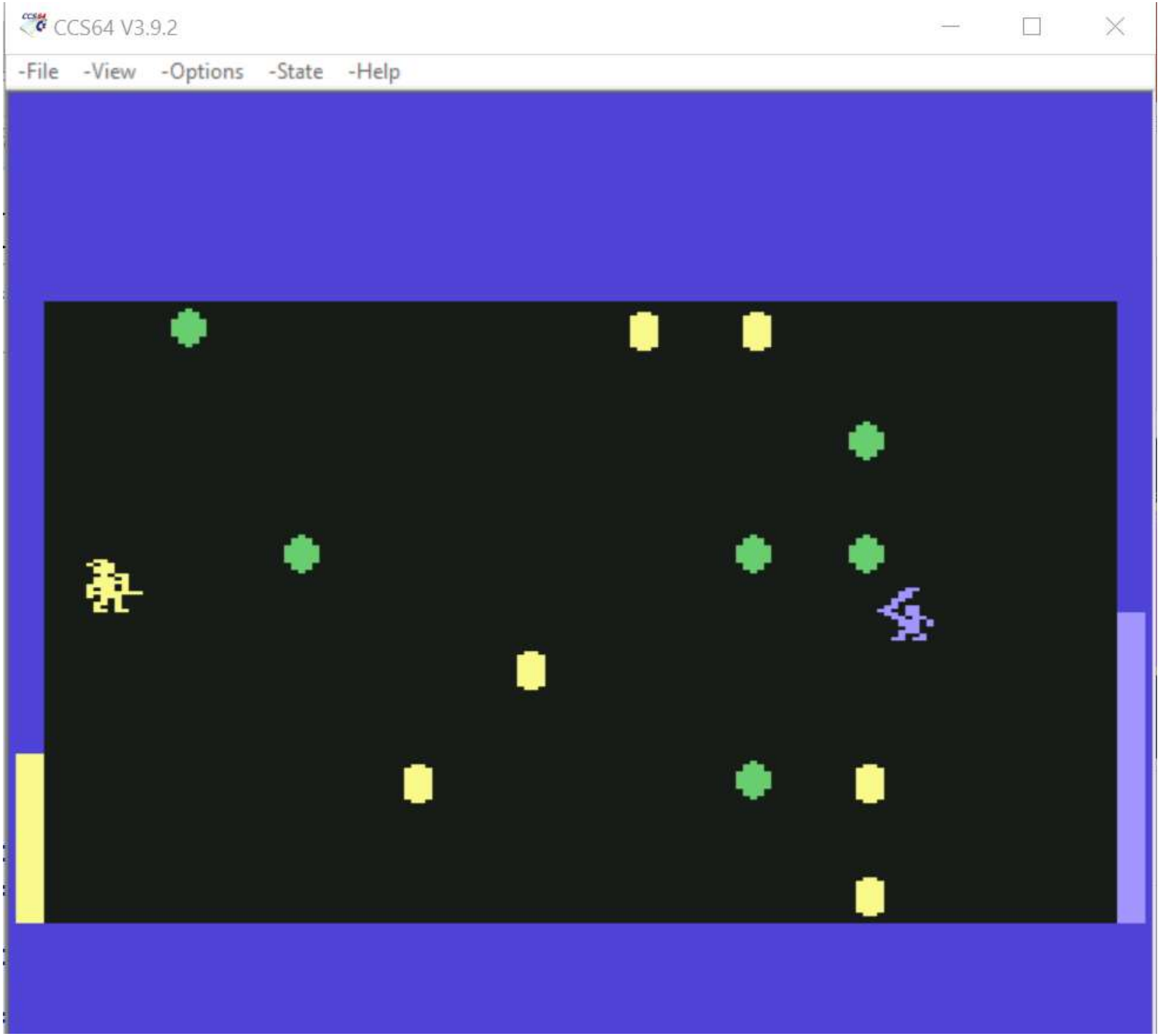
Related Work



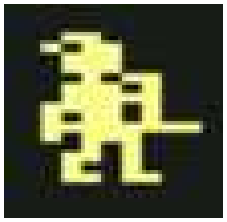
Playing Atari with Deep Reinforcement Learning

- Remarkable results by DeepMind
- Demonstrated how a computer learned to play Atari 2600 video games by observing just the screen pixels and receiving a reward when the game score increased
- Gave state-of-the-art results in six of the seven games it was tested on
- No adjustment of the architecture or hyperparameters.

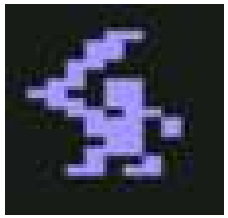

CCS64 V3.9.2
-File
-View
-Options
-State
-Help
ARCHON


CCS64 V3.9.2
-File
-View
-Options
-State
-Help

Archon



Knight



Goblin

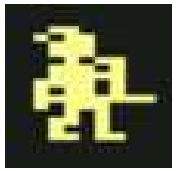
The Light and the Dark

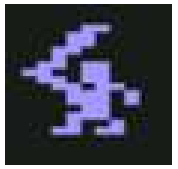
- Classic video game, developed in 1983 by Free Fall Associates
- Similar to chess, apart from the fight mode
- Generally in combat, a stronger player defeats a weaker player in either defending or capturing a square

Libraries used

For image processing

- OpenCV-Python
  - aimed at real-time computer vision
- Template matching
  - technique for finding areas of an image that match (are similar) to a template image (patch).
  - **Source image:** image in which we expect to find a match to the template image
  - **Template image:** patch image that will be compared to the template image





Libraries used

For automating the bot

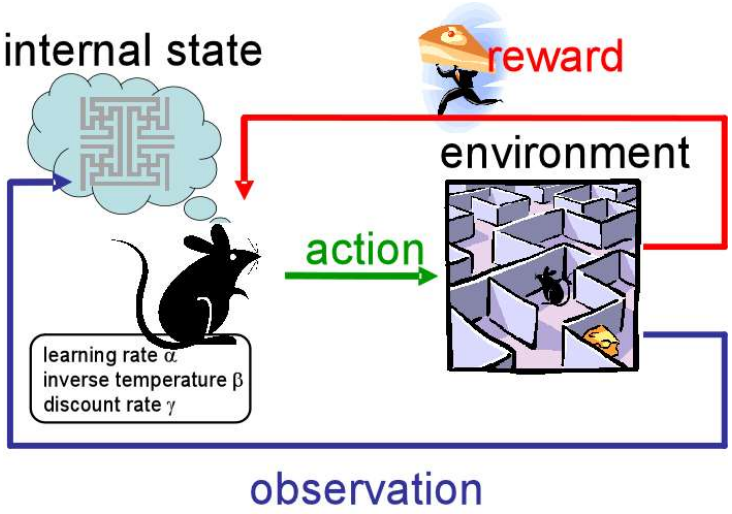
- PyautoGUI

- Keyboard

- PIL (Python Imaging Library)

SJSU SAN JOSÉ STATE UNIVERSITY

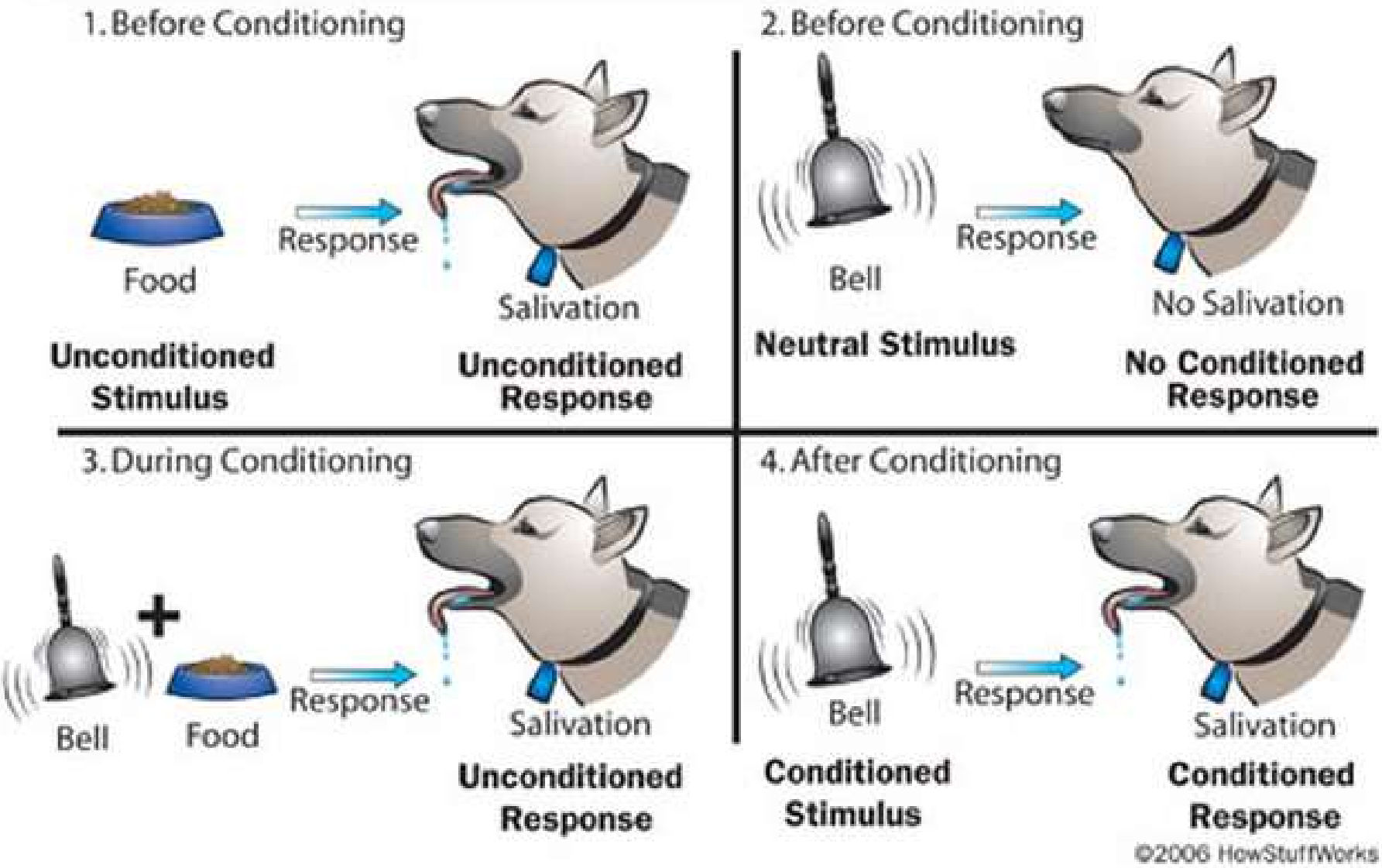
Introduction



Reinforcement Learning

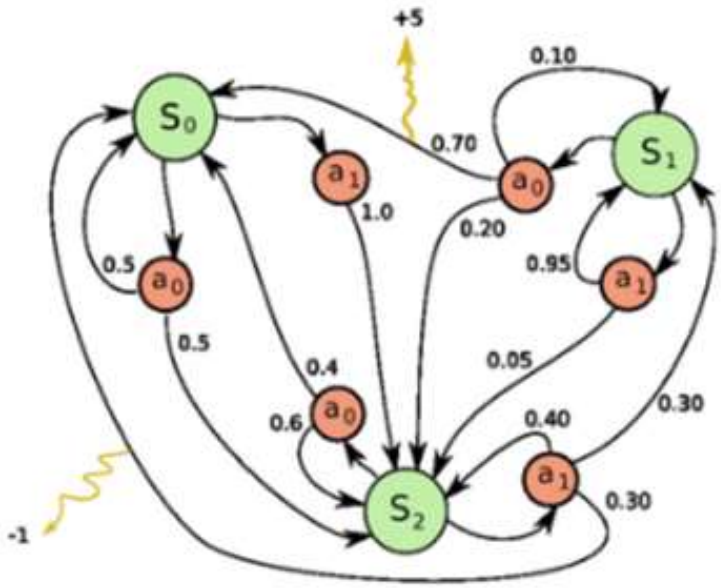
- General purpose framework for decision-making
- Inspired by behaviorist psychology
- Between supervised and unsupervised learning
- Agent learns to interact with an environment through occasional feedback
- The reward, to achieve a long-term goal.
- Each action by the agent influences it's future state

How Dog Training Works



Markov Decision Process

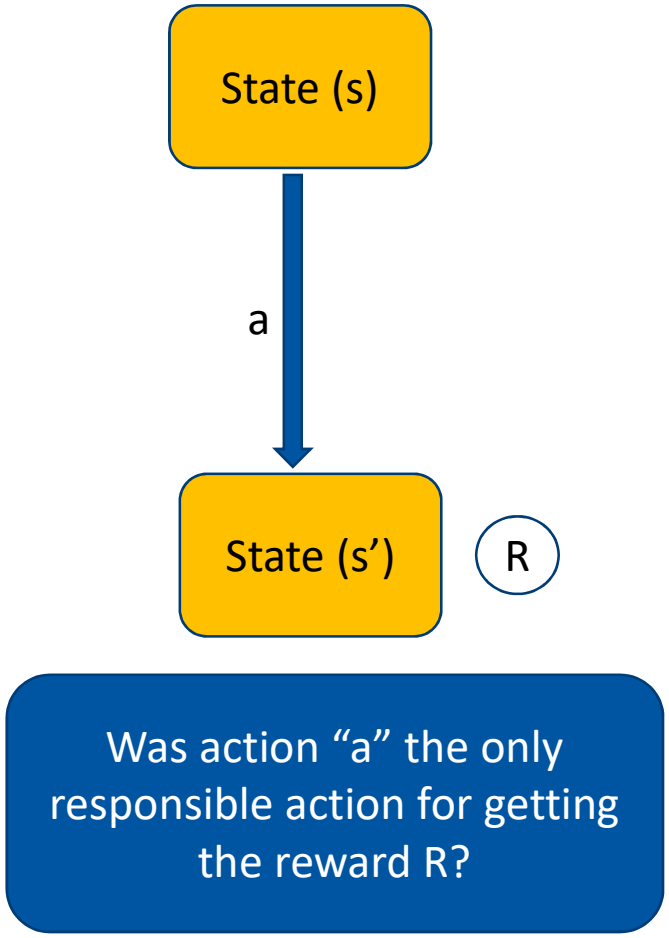
- Framework for modeling decision making situations
- Outcomes are partly random and partly under the control of an agent
- Set of states and actions, together with rules for transitioning from one state to another, make up a Markov decision process
- RL problem represented as a Markov Decision Process



Credit Assignment Problem

The problem?

- Agent performs an action and receives a positive reward
- No relation with actions performed just before getting that reward
- Q-learning propagates rewards back in time, until it reaches the crucial decision point which was the actual cause for the obtained reward.



Discounted Future Reward

- To perform well in the long-term, we need to take into account not only the immediate rewards, but also the future rewards
- Our environment is stochastic
- The more we dive into the future, the more it may diverge
- Discounted future reward:

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} \ldots + \gamma^{n-t} r_n$$

Exploration-Exploitation

The problem?

- An action plan to obtain certain rewards has been figured out.

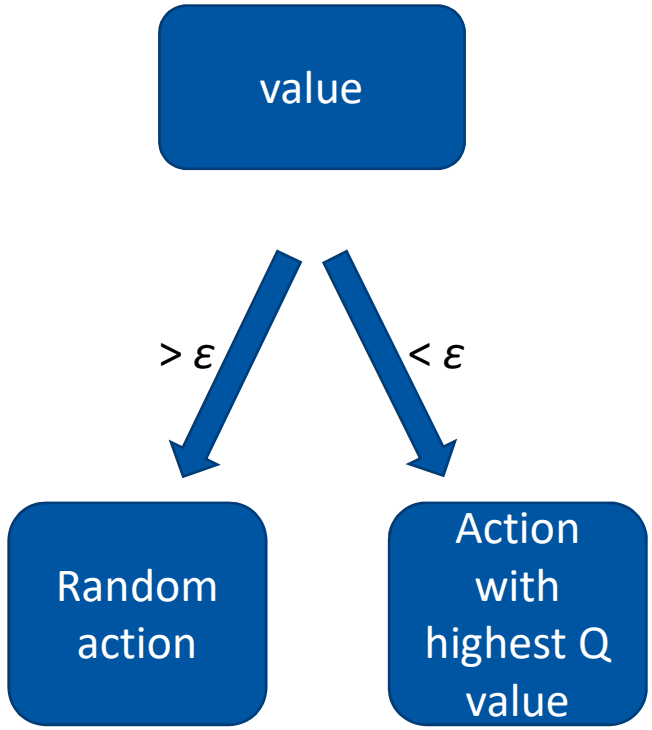
- Is it feasible to keep following the same action plan or experiment with something new that could result in a larger reward?

Exploration-Exploitation

ε-greedy exploration

- Agent takes the estimated optimal action (with the highest Q-value) most of the time and a random action with probability ε

- This ensures that the bot explores the search space and sees how actions not currently considered optimal would have fared instead



Components of RL

- Policy
  - Rule how we choose an action in each state
  - Maps from state to action (a = π(s))

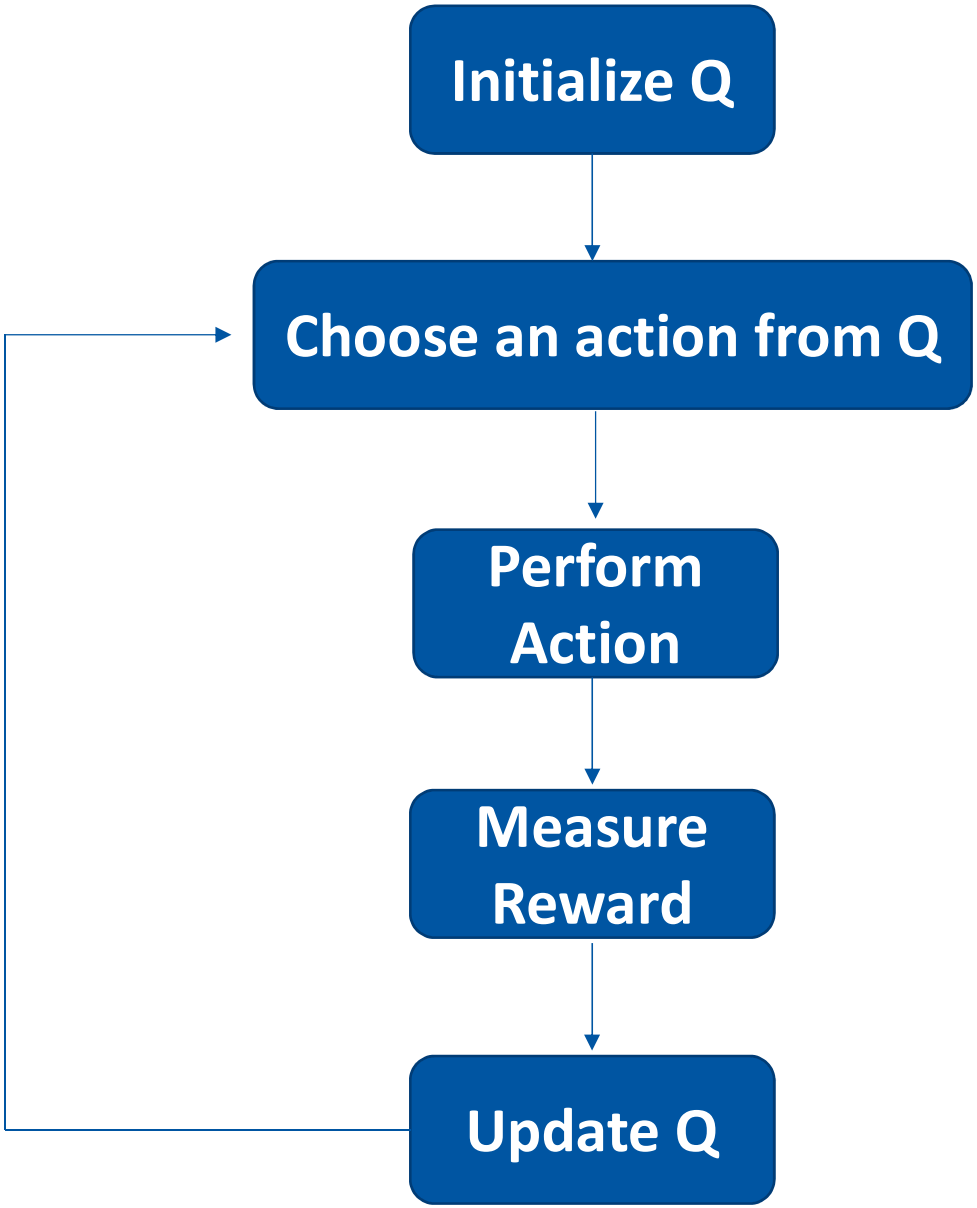
$$\pi(s) = argmax_a\, Q(s,a)$$

- Value Function
  - $Q(s_t, a_t) = \max R_{t+1}$
  - Best possible score at the end of the game after performing action a in state s
  - Prediction of future reward
  - Evaluate state and select between actions
- Model
  - Agent's representation of the environment

Q learning

- Model-free reinforcement learning technique
- Consists of an agent, states S and a set of actions per state A
- Agent moves from s -> s' by performing $a \in A$
- Executing an action in a specific state provides the agent with a reward
- Goal of the agent is to maximize its total reward (by learning the optimal action for each state)
- Optimal action has highest-long term reward R
- R: weighted sum of the expected values of all future steps staring from the current state

Q-learning Algorithm

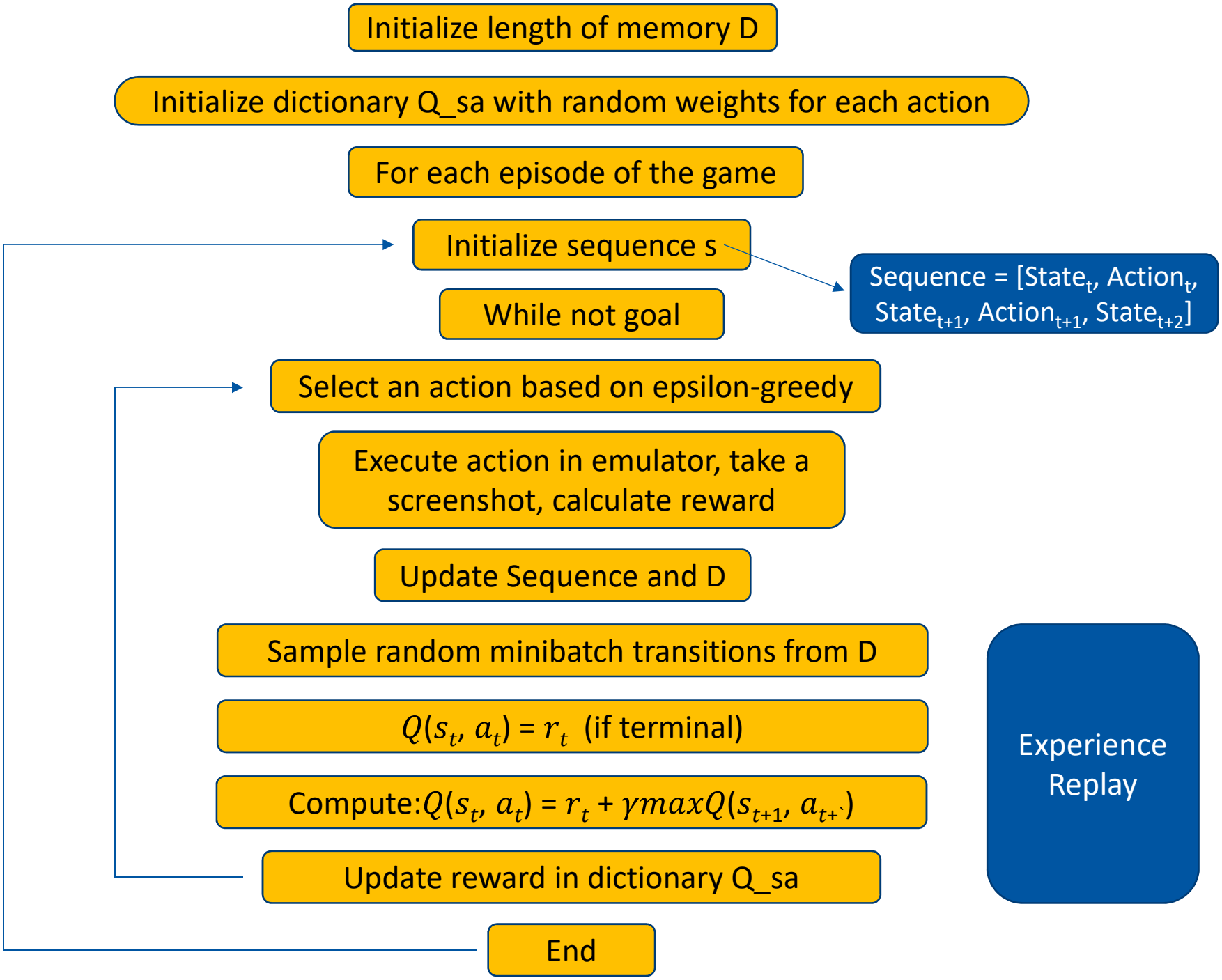


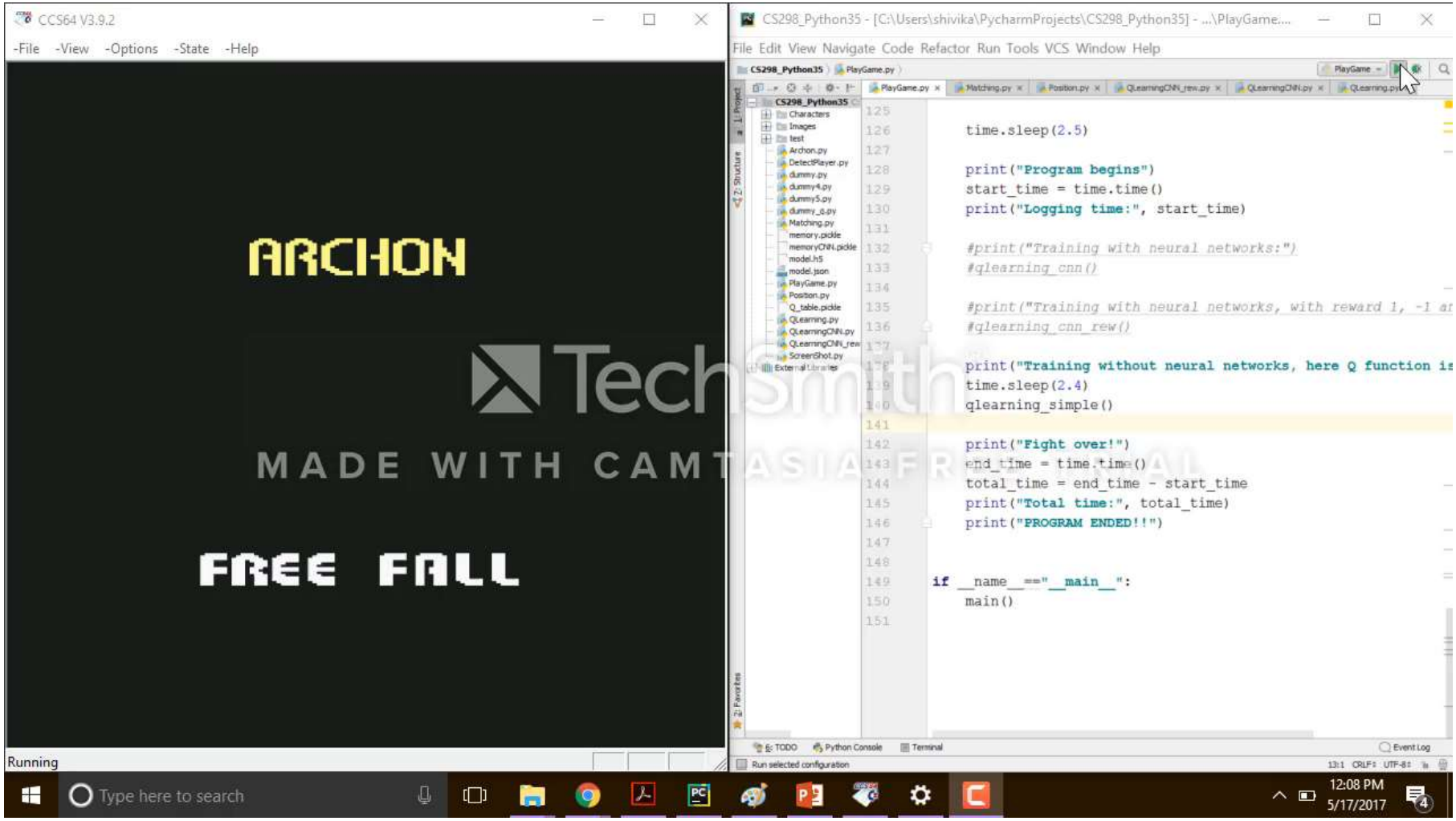
Without any form of training

Approach 1

States	Up A1	Down A2	Left A3	Right A4	Fire A5	Stay A6
(Pl1x, Pl1y), (Pl2x, Pl2y)	0.92	0.65	0.77	0.83	0.69	0.81
S2	0.83	0.59	0.79	0.93	0.61	0.51
S3	0.62	0.85	0.97	0.73	0.68	0.55

This action has the highest reward, hence action A1 will be chosen


Initialize length of memory D
Initialize dictionary Q_sa with random weights for each action
For each episode of the game
Initialize sequence s
Sequence = [State$_t$, Action$_t$, State$_{t+1}$, Action$_{t+1}$, State$_{t+2}$]
While not goal
Select an action based on epsilon-greedy
Execute action in emulator, take a screenshot, calculate reward
Update Sequence and D
Sample random minibatch transitions from D
$Q(s_t, a_t) = r_t$ (if terminal)
Compute: $Q(s_t, a_t) = r_t + \gamma max Q(s_{t+1}, a_{t+`})$
Update reward in dictionary Q_sa
End
Experience Replay

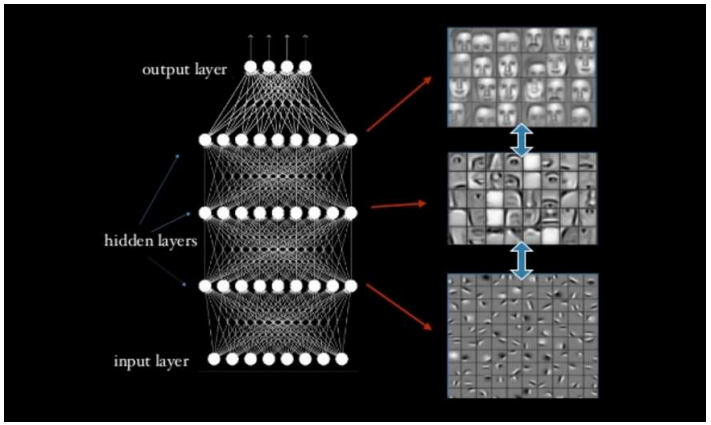


SJSU SAN JOSÉ STATE UNIVERSITY

Approach 2

Deep Neural Network

- Uses a cascade of many layers of nonlinear processing units for feature extraction and transformation



- Input of the algorithm is raw pixels

- Works using minimal domain knowledge

Convolutional Neural Network

- Act similarly to human receptive fields.
- Make sense of the game's screen output in a way that is similar to how humans are able to
- Several layers of convolutions with nonlinear activation functions applied to the results

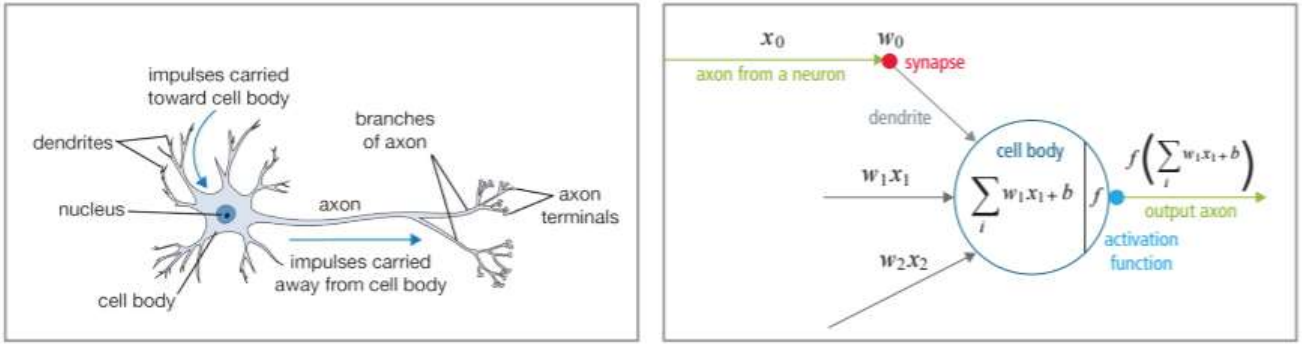
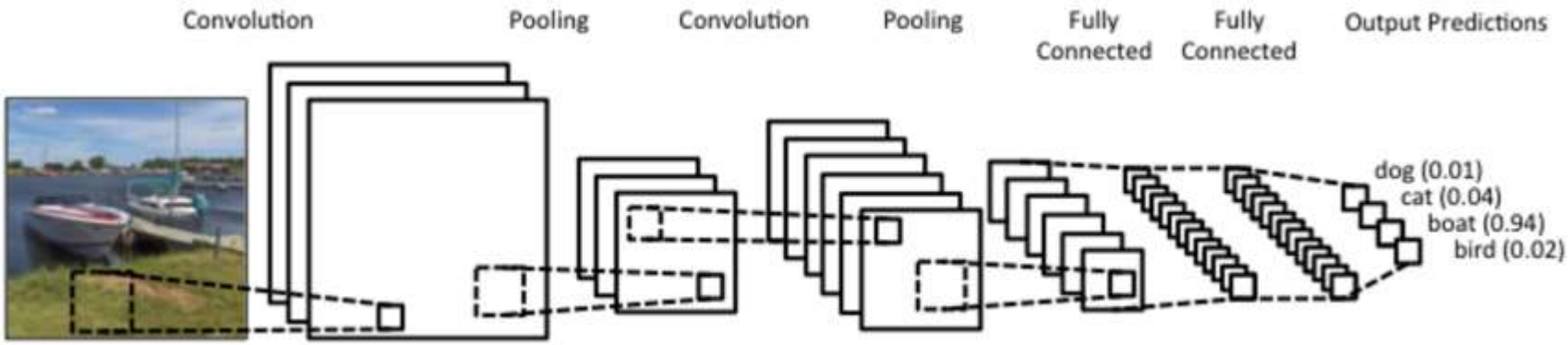


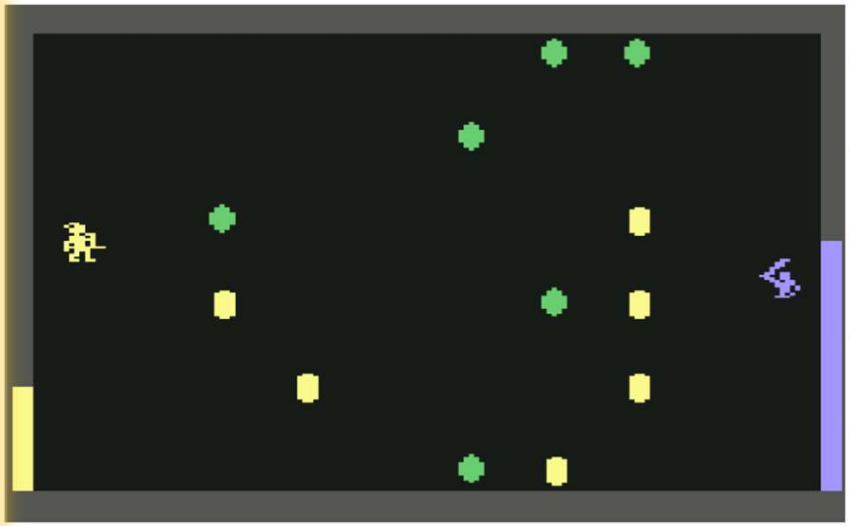
Illustration of a biological neuron (left) and its mathematical model (right)

CNN



- Sliding window function applied to a matrix
- Instead of considering each pixel independently, convolutional layers allow us to consider regions of an image
- Build edges from pixels, shapes from edges, and more complex objects from shapes..

OUR DEEP LEARNING MODEL



Input Image

convolutional neural net

Up

Down

Left

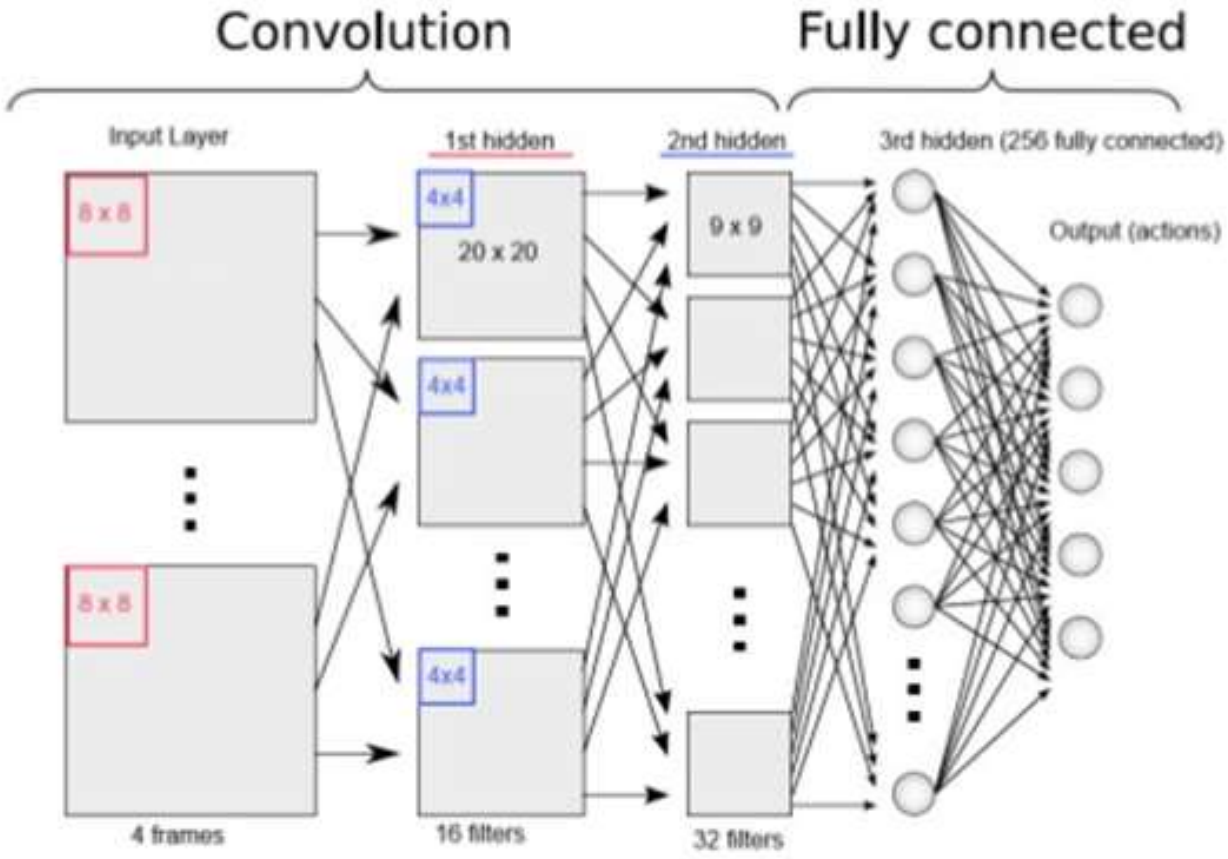
Right

Fire

Stay

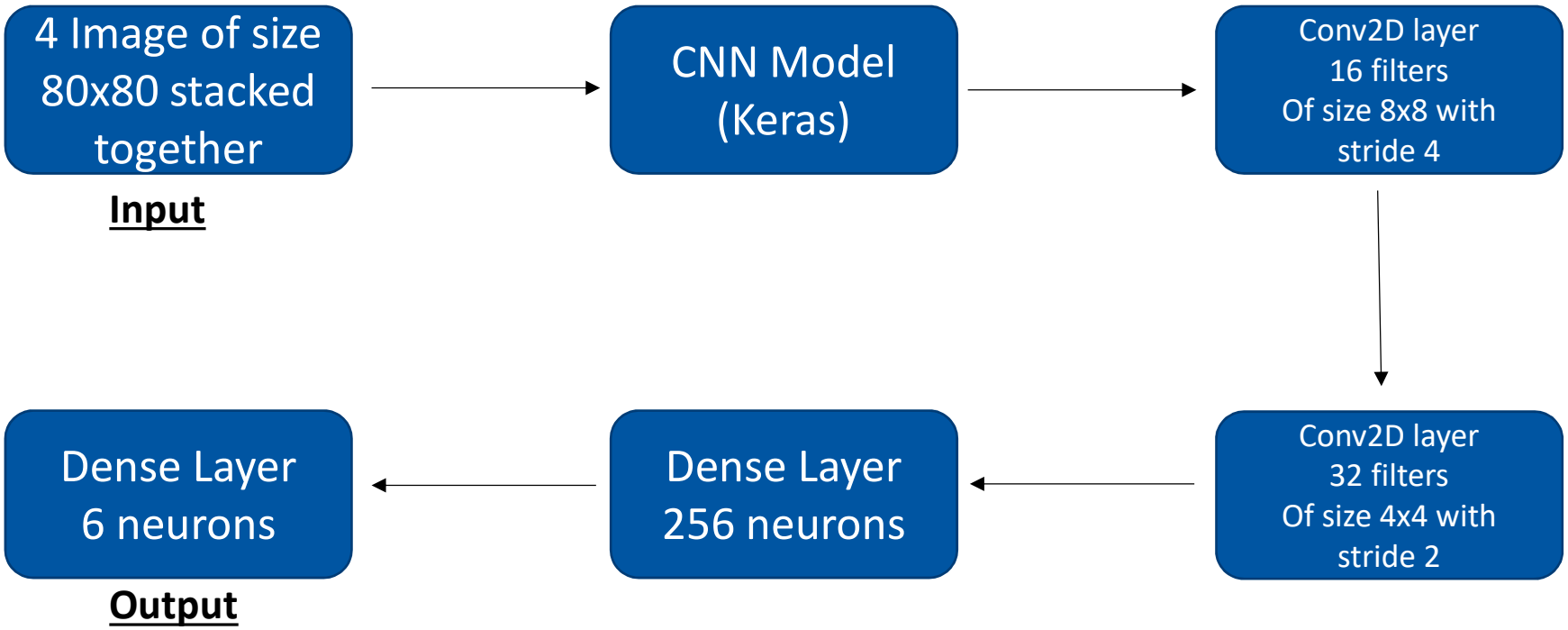
Possible Actions

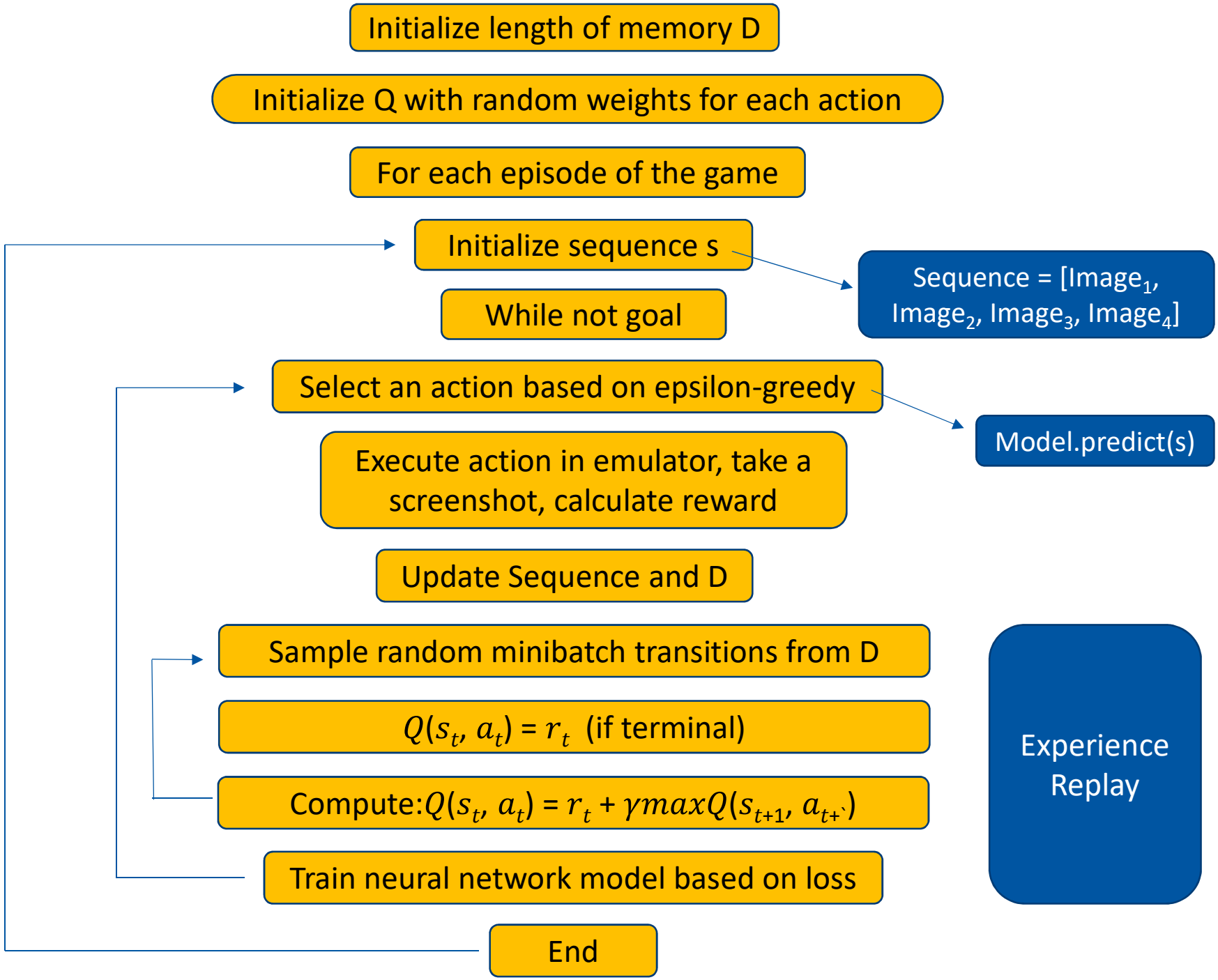
Network architecture

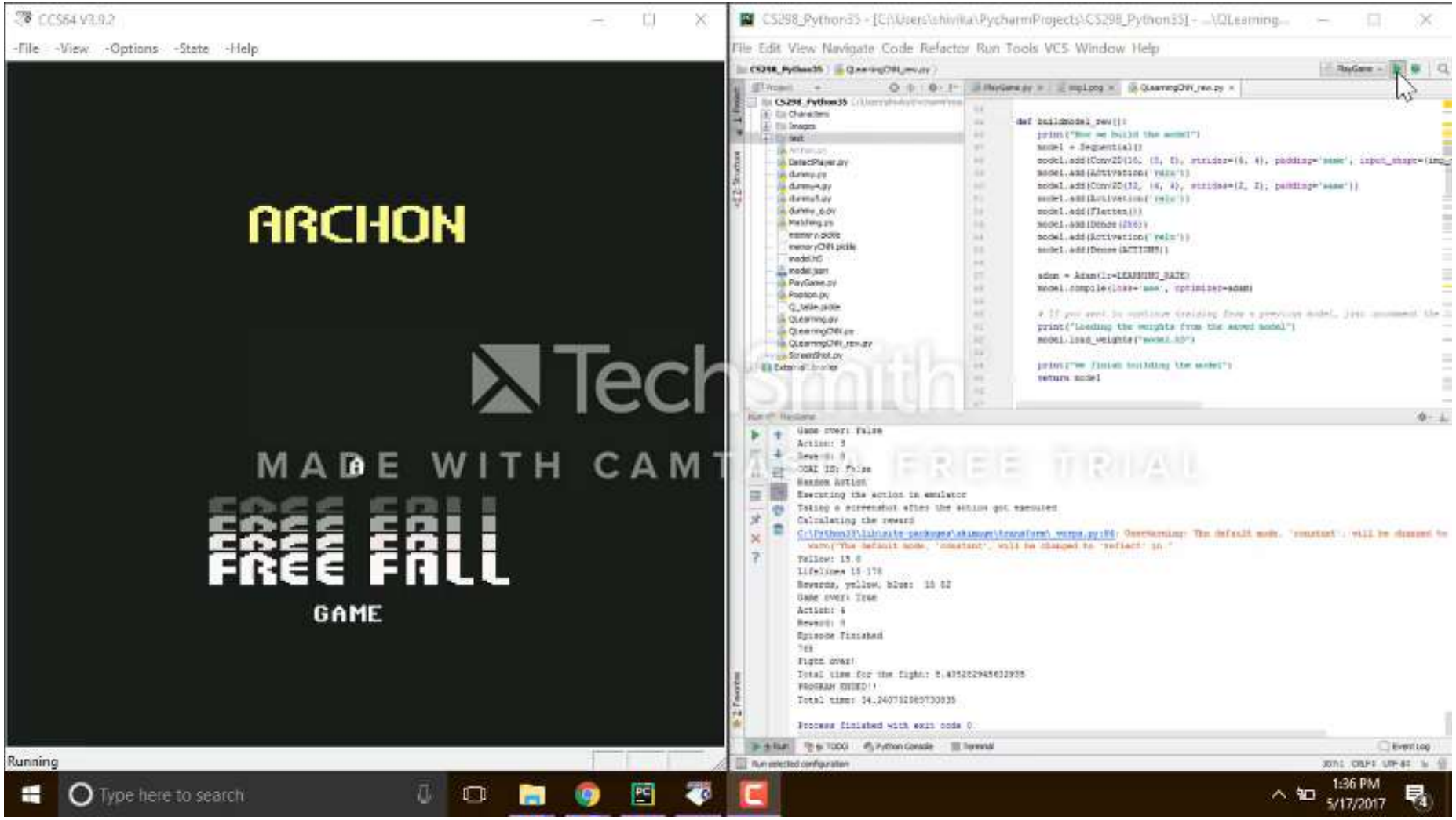


SJSU SAN JOSÉ STATE UNIVERSITY

Network Architecture

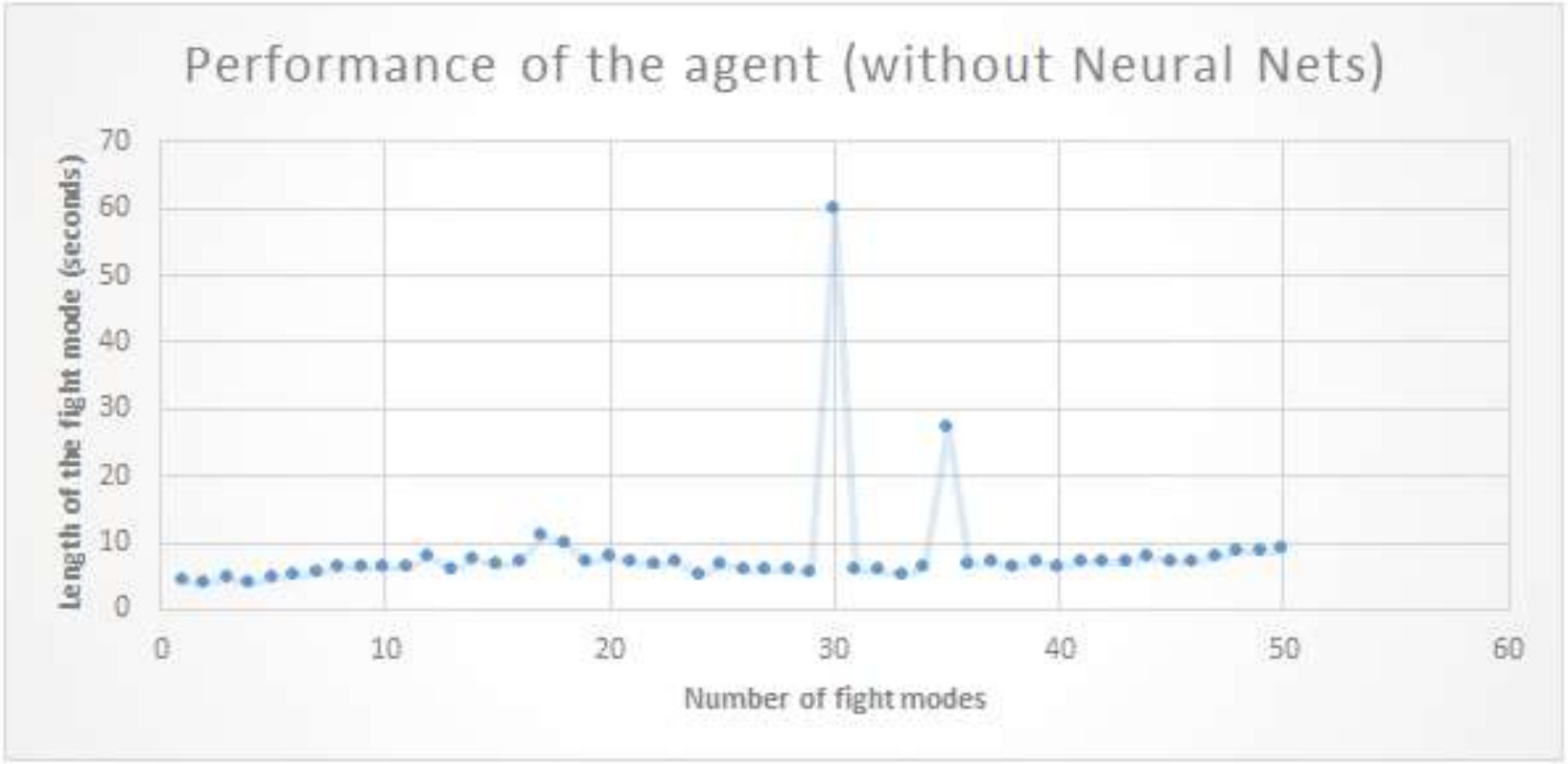



Initialize length of memory D
Initialize Q with random weights for each action
For each episode of the game
Initialize sequence s
Sequence = [Image1, Image2, Image3, Image4]
While not goal
Select an action based on epsilon-greedy
Model.predict(s)
Execute action in emulator, take a screenshot, calculate reward
Update Sequence and D
Sample random minibatch transitions from D
Experience Replay
$Q(s_t, a_t) = r_t$ (if terminal)
Compute: $Q(s_t, a_t) = r_t + \gamma max Q(s_{t+1}, a_{t+})$
Train neural network model based on loss
End

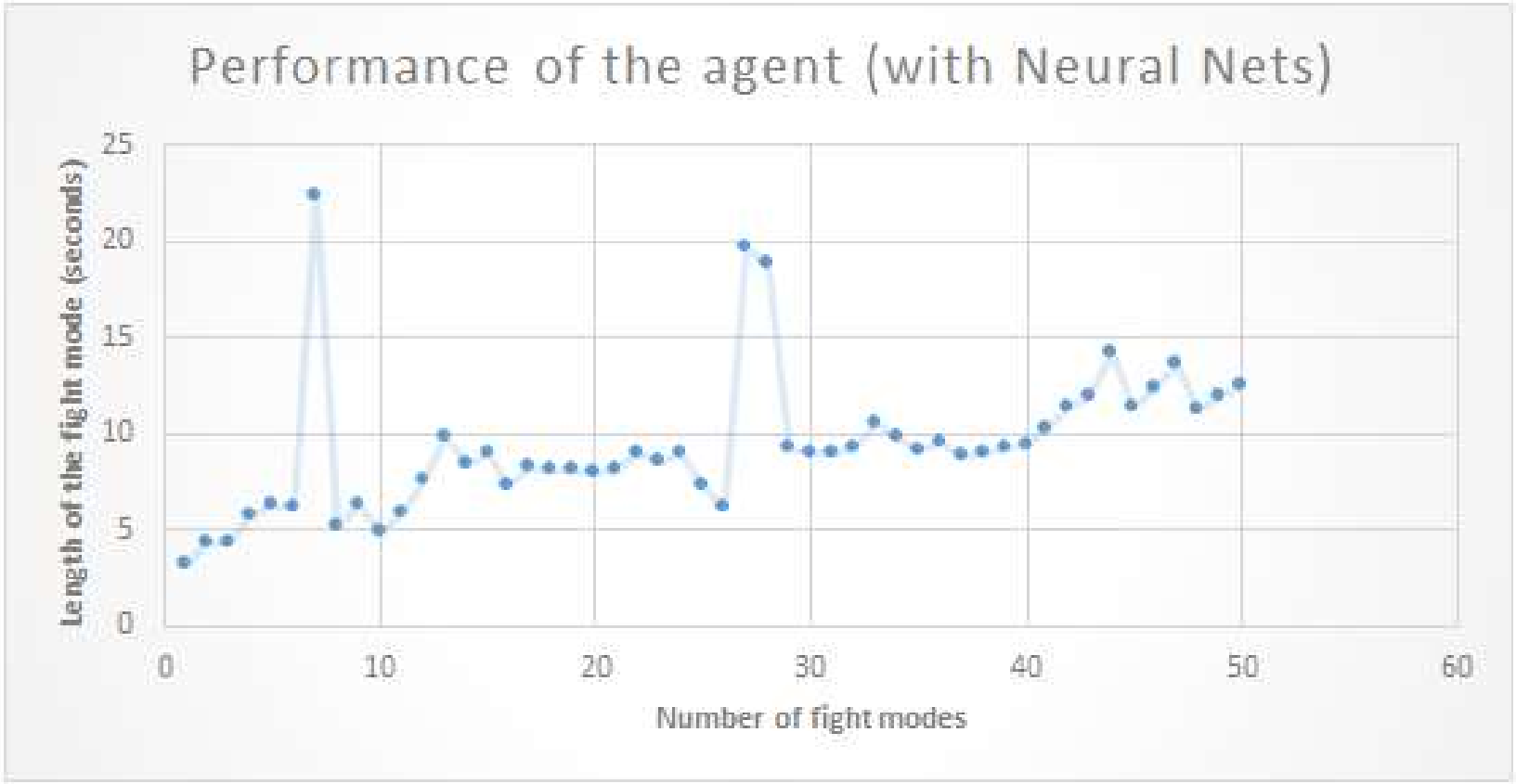

CCS64 V3.9.2
-File -View -Options -State -Help
ARCHON
FREE FALL
FREE FALL
FREE FALL
GAME
Running
Type here to search
CS298_Python35 - [C:\Users\shivika\PycharmProjects\CS298_Python35] - ...\QLearning...
File Edit View Navigate Code Refactor Run Tools VCS Window Help
def buildmodel_new():
print("Now we build the model")
model = Sequential()
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dense(ACTIONS))
print("We finish building the model")
return model
Game over: False
Action: 3
Random Action
Executing the action in emulator
Taking a screenshot after the action got executed
Calculating the reward
Yellow: 15 0
Lifetime: 15 170
Rewards, yellow, blue: 15 82
Game over: True
Action: 4
Reward: 0
Episode finished
Flight over!
PROGRAM ENDED!!
Process finished with exit code 0
1:36 PM
5/17/2017

Result

- Experiments were performed on 50 iterations of the game, for two different policies.
- **First approach**:
  - Agent starts playing better gradually, but its improvement is very slow.
  - Survives the fight for approximately 5 more seconds.
- S**econd approach**:
  - Agent learns comparatively faster with neural network as compared to when trained without neural networks.
  - Survives the game for about 8 more seconds on the 50th iteration of the game.



SJSU SAN JOSÉ STATE UNIVERSITY



SJSU SAN JOSÉ STATE UNIVERSITY

Conclusion

- This research demonstrated that reinforcement learning, when implemented with neural networks, perform a better and faster training of the bot as compared to when the Q function is used as a table.

- We trained an artificially intelligent bot to play Archon using two different policies

System Configuration

Windows edition

Windows 10 Home

© 2016 Microsoft Corporation.
All rights reserved.

Windows 10

System

Processor:	Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz 2.40 GHz
Installed memory (RAM):	16.0 GB
System type:	64-bit Operating System, x64-based processor

Lenovo

SJSU SAN JOSÉ STATE UNIVERSITY

Future Scope

- Train using a GPU and thus compare performance
- Train the bot to fight players other than the Knight (light side) and Golem (dark side)
- Train the bot to play the game from the dark side
- Apply the same technique to the strategy mode of the game and finish the game

SJSU SAN JOSÉ STATE UNIVERSITY

References

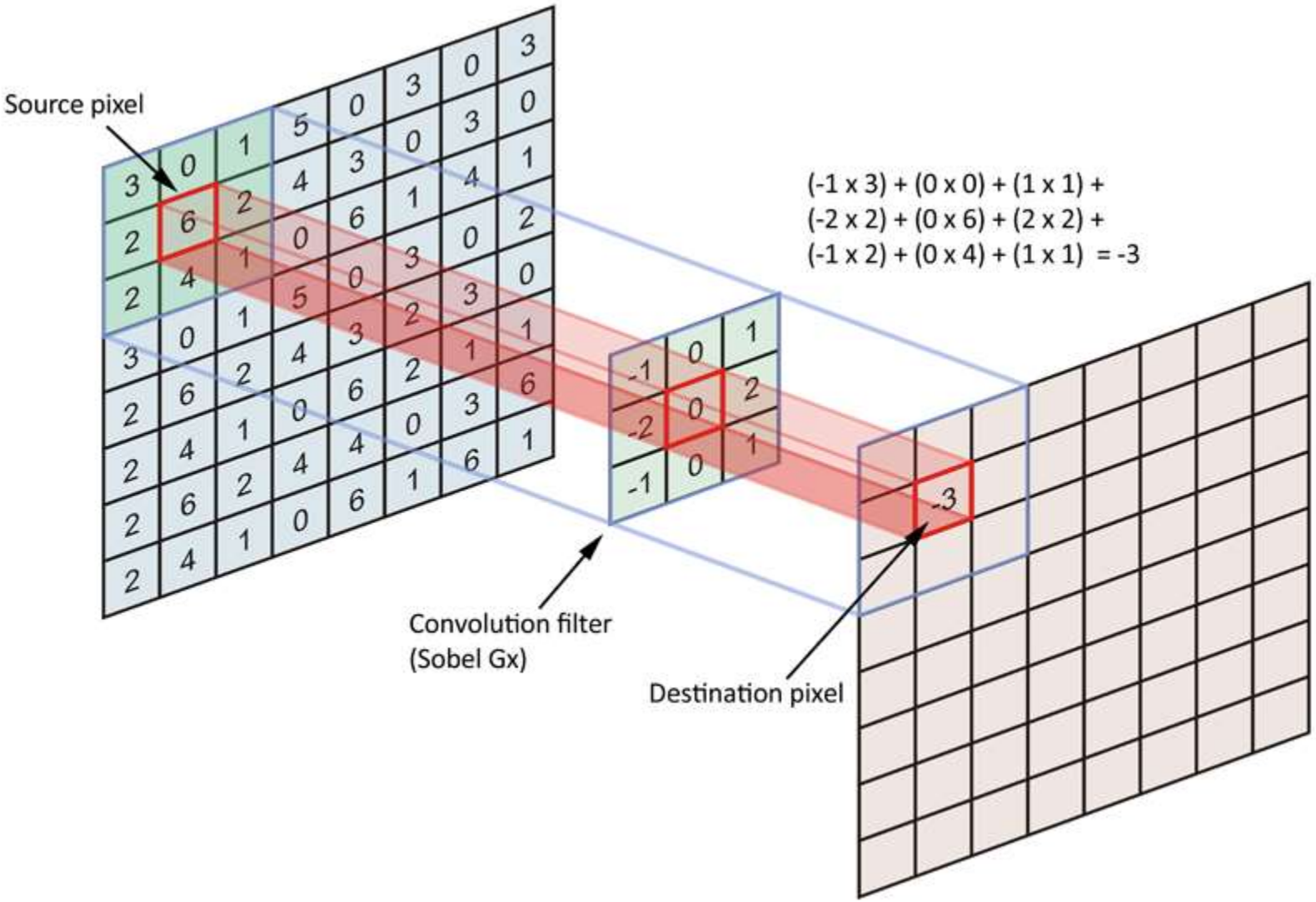
- https://ip.cadence.com/uploads/901/cnn_wp-pdf
- http://www.cs.umd.edu/~djacobs/CMSC733/CNN.pdf
- https://www.slideshare.net/roelofp/python-for-image-understanding-deep-learning-with-convolutional-neural-nets
- https://en.wikipedia.org/wiki/Q-learning
- https://www.slideshare.net/nikolaypavlov/deep-qlearning
- http://cs231n.stanford.edu/reports/2016/pdfs/121_Report.pdf
- http://www.krigolsonteaching.com/reinforcement-learning.html

Thank you.

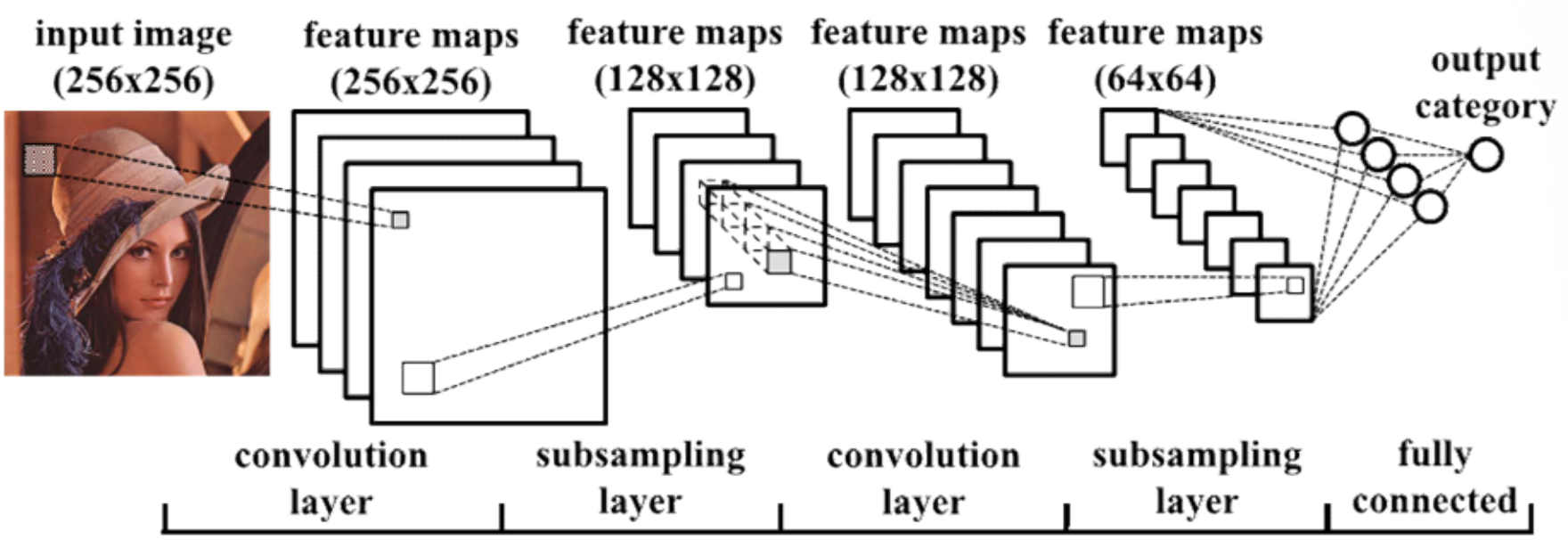
SJSU SAN JOSÉ STATE UNIVERSITY

Commodore 64

- 8-bit home computer introduced in January 1982 by Commodore International
- Emulator from http://www.ccs64.com
- Low resolution, less pixels, easier for image processing techniques



SJSU SAN JOSÉ STATE UNIVERSITY


input image
(256x256)
feature maps
(256x256)
feature maps
(128x128)
feature maps
(128x128)
feature maps
(64x64)
output
category
convolution
layer
subsampling
layer
convolution
layer
subsampling
layer
fully
connected

Appendix

Q-learning Pseudocode

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
    Initialize s
    Repeat (for each step of episode):
        Choose a from s using policy derived from Q (e.g., ε-greedy)
        Take action a, observe r, s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha\left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]$
        $s \leftarrow s'$;
    until s is terminal

Pseudocode

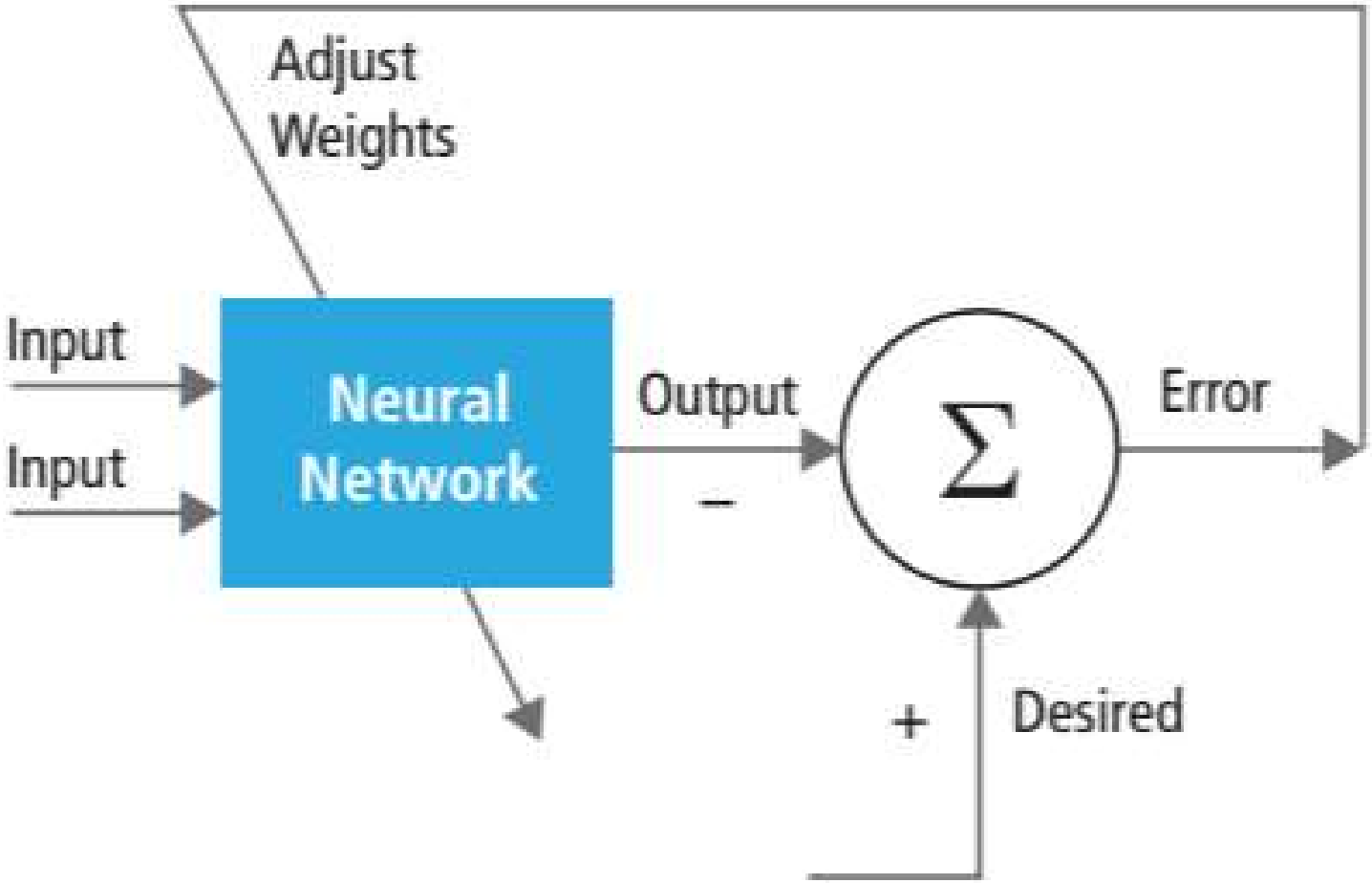
Algorithm 1 Q function as a table

```
procedure QLEARN
    Initialize replay memory D to capacity N.
    Initialize Q with random weights for each action
    for each episode of game do
        seq ← [s_t, a_t, s_t, a_t, s_t]
        while not goal do
            Based on the value of epsilon
            either select a random action
            or select action with maximum weight
            Execute action in emulator
            Take a screenshot
            Calculate the reward
            Calculate the state
            Update seq ← [s_{t+1}, a_{t+2}, s_{t+2}, a_{t+3}, s_{t+3}]
            D ← [s_t, a_t, r_t, s_{t+1}]
            Sample random minibatch transitions from D
            Compute: Q(s_t, a_t) = r_t + γ max Q(s_{t+1}, a_{t+1})
            s_{t+1} ← s_t
        End while
    End for
```

Pseudocode

Algorithm 2 Deep Q learning algorithm

```
procedure QLEARN
    Initialize replay memory D to capacity N.
    Initialize Q with random weights for each action
    for each episode of game do
        s_t ← image
        seq ← [s_t, s_t, s_t, s_t]
        while not goal do
            Based on the value of epsilon
            either select a random action
            or select action with maximum weight
            Execute action in emulator
            Take a screenshot
            Calculate the reward
            Calculate the state
            Update seq ← [s_t, s_t, s_t, s_t, s_{t+1}]
            D ← [s_t, a_t, r_t, s_{t+1}]
            Sample random minibatch transitions from D
            Predict Q values from the neural network model
            Compute: Q(s_t, a_t) = r_t + γ max Q(s_{t+1}, a_{t+1})
            s_{t+1} ← s_t
            Calculate loss to update weights
        End while
    End for
```



Training of neural networks

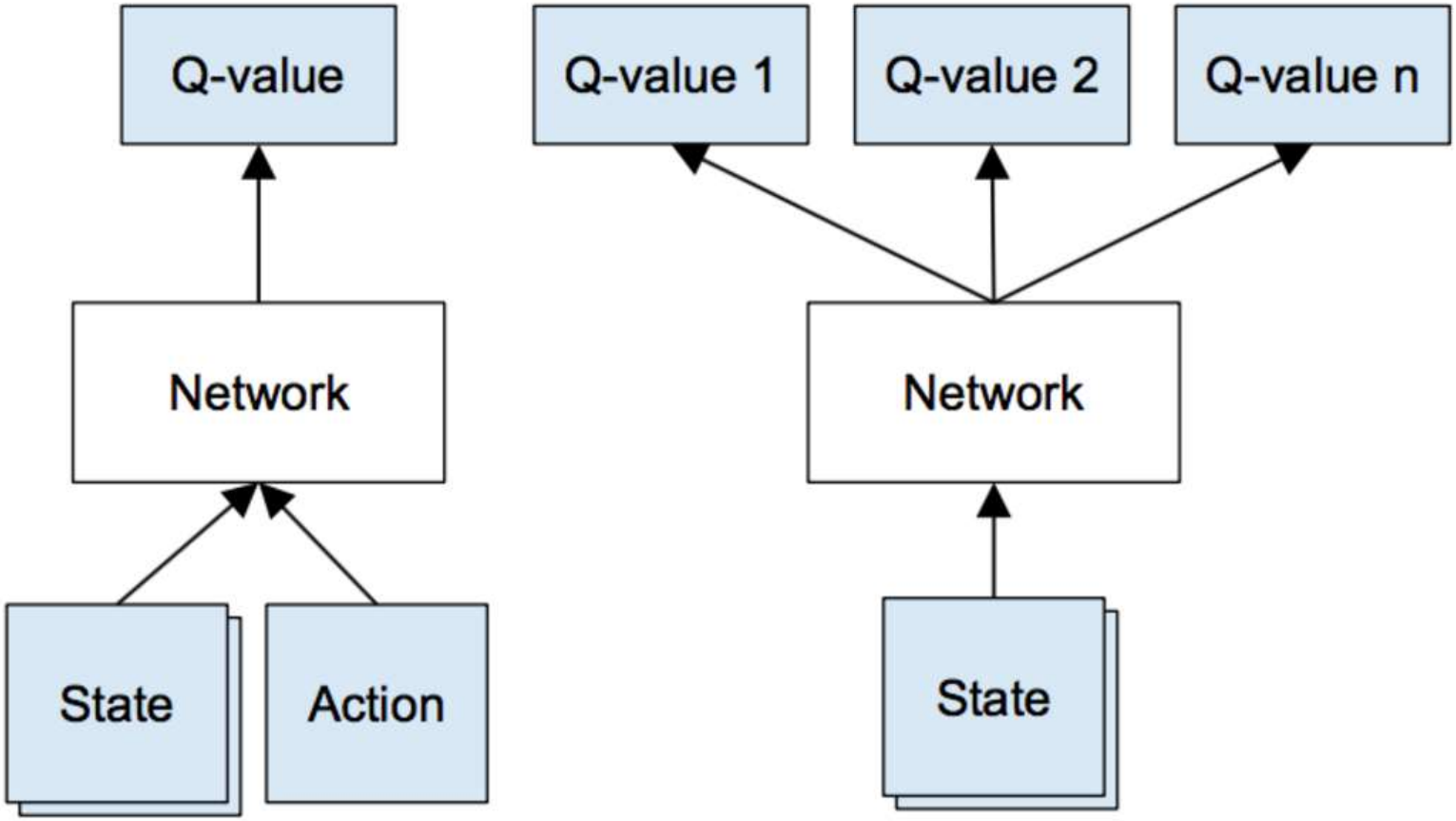


Figure 3: *Left:* Naive formulation of deep Q-network. *Right:* More optimized architecture of deep Q-network, used in DeepMind paper.